Chapter 6

Syntactic Realization

6.1 Introduction

Syntactic realization concerns the translation of phrase specifications into grammatical expressions in the target language. The complexity of this process mainly depends on the abstraction of the phrase specifications. In more concrete representations (e.g. strings of canned text), syntactic realization amounts to little more than string concatenation and some orthographic processing. If, on the other hand, the representation is very abstract (e.g. a predicate-logical expression), the process involves a substantial amount of work. Following the standard architecture for NLG systems (Section 5.2.1), we assume that lexical choice, aggregation, and referring expression generation belong to microplanning rather than to syntactic realization. As a consequence, our phrase specifications already contain the lemmas of content words, an abstract hierarchical specification of the syntactic form, and features indicating the type of referring expression. These Abstract Phrase Specifications (APS's) are somewhere in the middle on the scale of abstractness. Some examples of APS's and their realizations are given in (1).

(1) a. APS: (sneltrein)
String: een sneltrein
a fast-train

b. APS: (sneltrein<def>)
String: de sneltrein
the fast-train

c. APS: (sneltrein<def>(P:naar(Amsterdam<prop>)))
String: de sneltrein naar Amsterdam
the fast-train to Amsterdam

d.	APS:	`(V:nemen[<listen>]` `[sneltrein<def>(P:naar(Amsterdam<prop>))])`
	String:	u neemt de sneltrein naar Amsterdam
		you take the fast-train to Amsterdam

As explained in Section 5.3.1, our goals regarding prosodic realization require the use of this fairly abstract representation, and the generation of a complete syntactic structure for each utterance. Hence, the task of the syntactic realization module of CONPAS is far from trivial. Syntactic realization of APS's involves basically three tasks: determining the word order, inflecting content words, and inserting function words. In addition, and specific to SLG, it requires the distribution of focus, and the expansion of numerals, times, dates, and abbreviations. Moreover, not only strings of words are generated, but also their syntactic structures, which are to be used for prosodic realization. These syntactic structures conform to the analysis of Dutch syntax outlined in Section 2.6, which was also used for the syntactic annotation of the flower descriptions in the previous two chapters. These structures may be less intricate than those assumed in current syntactic models, but they proved to be sufficient for our purposes. That is, they serve as convenient data structures in the process of syntactic realization (e.g. to enforce agreement relations), and they provide the necessary information for prosodic realization. The results of the current work can therefore be incorporated in other syntactic frameworks without much difficulty.

We have chosen to split the realization process into three natural steps. Accordingly, the syntactic realization module SynR is divided into three components:

1. Base, which is primarily concerned with expanding an APS to a full-blown syntactic structure that adheres to the principles of $\overline{X}$-theory

2. Trans, which is responsible for enforcing grammatical constraints, determining the word order, and some additional tasks like distributing focus

3. Flex, which inserts function words and inflects content words

Like CONPAS itself, these components form a strict pipeline.

The three components are described below, including an introduction to the grammar formalisms used for their implementation. The goal is to provide a description that highlights interesting aspects, in particular with respect to SLG, at the expense of technical details of the implementation. The chapter closes with a discussion and summary of the results.

6.2 The **Base** component

6.2.1 The format of Abstract Phrase Specifications

An APS is a tree structure in which every node has the following form:

(2) **Format of an APS node**
(*category* : *lemma* < *features* > *daughters*)

The *category* is a major or a minor syntactic category. There are six major categories: N (Noun), V (Verb), A (Adjective), ADV (Adverb), P (Preposition), and NUM (Numeral). The minor categories are: D (Determiner), C (Complementizer), CONJ (Conjunction[1]), LET (Letter[2]). Although the major-minor distinction is irrelevant to the format of an APS itself, it is used during syntactic realization: following Gazdar, Klein, Pullum, and Sag (1985), only major categories project a phrase (e.g. N triggers the generation of an NP). This will be further explained in Section 6.2.3.1.

A *lemma* is the base form of a word, as used in a normal dictionary. In general, only the content words in an APS have a lemma, whereas function words have none (i.e. the empty string). However, it is sometimes necessary to overrule the default realization of a function word by explicitly prescribing its lemma. For instance, `(CONJ:alsmede)` forces the use of *alsmede* 'as well as' rather than the default *en* 'and'.

With respect to prepositions, a distinction is assumed between purely syntactic prepositions subcategorized for by a verb, and meaningful prepositions, e.g. indicating a spatial relation. The former are automatically inserted by SynR, the latter have to be supplied in the APS. A lemma in an APS may also be empty, which is often the case for nouns. Dependent on the features supplied, SynR generates an appropriate word form, e.g. a demonstrative determiner.

The *features* form a comma-separated list of semantic and syntactic features. In general, features mark exceptions to the default. For instance, nouns are assumed to be singular, unless they bear the feature `plur`; verbs are assumed to carry present tense, unless the feature `past` is supplied.

The *daughters* form a list of nodes, which makes the definition of a node recursive. Each node can have an arbitrary number of daughters, including zero. The mutual order of the daughters is irrelevant, with two exceptions. First, arguments are assumed to be ordered, in order to facilitate linking of arguments to syntactic functions (Section 6.3.3). Second, adjectives surface in the same order they occur in the APS (Section 6.3.7.5).

In an SLG environment primarily targeted at research, it is important that APS's can be easily read and written. This is partly accomplished by using defaults, which relieve the text planner (or the user) of the necessity to specify

[1]The syntactic category of all words that can be used to connect the conjuncts in a coordination. This includes a disjunction like *of* 'or'. We have no account yet for coordination with a 'split conjunction' like *zowel X als Y* 'both X and Y'.

[2]For example, the category of the letter *A* in *perron 1A* 'platform 1A'.

all features. In addition, there are some notational conventions ('syntactic sugar'):

- If the category and the semicolon are omitted, the node will interpreted as a noun. That is, `(trein)` is equivalent to `(N:trein)`
- If a noun starts with a capital, it will be interpreted as a proper noun, i.e. `(Amsterdam)` is equivalent to `(Amsterdam<prop>)`.
- Rather than using `arg` to indicate that a daughter functions as an argument, it can be enclosed by square brackets (`[...]`) instead of parentheses.
- An empty list of features (`<>`) may be omitted.
- Braces ({})can be used to indicate a coordinate structure. For example, `(N:{(N:stoptrein)(N:sneltrein)})` is equivalent to `(N:<conj>(N:stoptrein)(N:sneltrein))`

As a result, we can write `[{(Utrecht)(Amsterdam)}]` rather than `(N:<arg,conj>(N:Utrecht<prop>)(N:Amsterdam<prop>))`.

6.2.2 The AGFL system

APS's can be formally defined by a context-free grammar. We can build a parser that recognises all and only APS expressions, but this is not particularly interesting from an NLG viewpoint. However, if during the parsing process the input can be *transduced* to a new structure, we can use this to map APS's onto natural language expressions. This is exactly what the AGFL system (Koster 1991; Derksen and Potjer 2000) allows us to do.

AGFL stands for *Affix Grammars over Finite Lattices*. It is a member of the family of *two-level grammars*. The first level consists of the context-free rewriting rules familiar to any linguist.

(3)
```
NP : Det, N.
```

The second level consists of context-free affix-rules, rewriting affix-nonterminals into affix-terminals or other affix-nonterminals.[3]

(4)
```
number :: sing; plur.
definiteness :: def; indef.
```

In contrast to the syntax-level, the affix-level does not allow recursion.

The two levels can be combined by using the affixes as *parameters* to nonterminals of the syntax rules. According to the *Consistent Substitution Rule*, all occurrences of an affix (with the same name) within a rule have the same value, which implies unification of the affix values. For instance, both occurrences of `number` will get the same value in the first rule the following grammar.

[3]The newest versions of AGFL use '—' rather than ';' to separate alternatives.

(5)
```
NP : Det(number,definiteness), N(number).

Det(sing,indef) :  "a".
Det(number,def) :  "the".
Det(plur,indef) :  .

N(sing) :  "flower".
N(plur) :  "flowers".
```

This grammar describes the noun phrases *a flower*, *the flower*, *flowers*, and *the flowers*, but rejects *a flowers* and *flower*.

The AGFL-system takes the grammar defined by (4) and (5), and generates a parser for it that produces a parse tree like (6) for the input *a flower*.

(6)
```
NP{Det(sing,indef){"a"},N(sing){"flower"}}
```

Of course, multiple trees will be generated if the input is ambiguous according to the grammar.

The affix mechanism is a powerful tool for making large grammars more compact. In addition, it allows a description that captures linguistic generalisations. It includes a number of special features that make it suitable for computational linguistic purposes. Particularly important for our purposes is that AGFL allows *transduction*: each rule can have a transduction part – starting with a forward slash (/) – that defines the string that is produced when the rule is successfully applied in the parsing process. Consider the grammar in (7).

(7)
```
NP : Det(number,definiteness), N(number) / Det, N.

Det(sing,indef) :  "a" / "een".
Det(number,def) :  "the" / "de".
Det(plur,indef) :  .

N(sing) :  "flower" / "bloem".
N(plur) :  "flowers" / "bloemen".
```

The AGFL-system can generate a transducer for this grammar. Given the inputs *a flower* and *the flowers*, the transducer will output the strings *een bloem* and *de bloemen* respectively. Thus, the transducer for grammar (7) is in fact a very primitive machine translation system. However, this approach is unlikely to be easily scaled up. In particular, the formalism is not very suitable for describing the complex word order phenomena in clauses (Section 6.3.7), or the long distance dependencies which may result from wh-movement. Moreover, since the descriptive power of an AGFL is that of a context-free rewrite grammar, accounting for certain types of dependencies may even prove to be theoretically impossible.

The same goes for using AGFL in a generation system: it is not suitable for mapping APS's onto grammatical expressions *directly*. However, this is not our goal, as we intend to build a modular generation system. We envision the generation process as naturally consisting of three steps. We therefore use the strengths of AGFL in the first and last steps, i.e. for mapping APS's onto basic syntactic structures in Base, and for inserting and inflecting words in Flex, but avoid its weakness by using a more powerful – but also slower and more complex – transformational grammar for the second step, i.e. for enforcing agreement relations and determining word order. Here, we continue with a description of the first step.

6.2.3 The tasks of Base

Base is the grammar that defines valid APS's.[4] At the same time, it defines the transduction of APS's into basic syntactic structures that can be processed by GRAMTSY in the next component. GRAMTSY, the interpreter for transformational grammars, will be described later on; for the moment, only the *format* of its syntactic structures is relevant. This is essentially the familiar labelled bracket structure enhanced with features.

(8) a. **Format of non-terminal node in GRAMTSY**
(*category* < *features* > - *daughters*)

b. **Format of a terminal node in GRAMTSY**
(*category* < *features* > - *lemma*)

Conceptually, mapping APS's onto basic syntactic structures involves the following tasks: (1) projecting the basic syntactic structure, (2) adding purely syntactic information, (3) adding the structure of compound words, and (4) expanding numbers and times.

6.2.3.1 Projecting the basic syntactic structure

Base projects a syntactic structure that adheres to a simplified version of $\overline{X}$ theory (Jackendoff 1977). It is assumed that every lexical item of a major category has a phrasal projection of the same category, i.e. an N is directly dominated by an NP, a V by a VP, etc. In contrast, minor categories have no phrasal projection. No intermediary levels of projection are assumed. The syntactic structures corresponding to the examples in (1) are given in (9).[5]

(9) a. APS: `(sneltrein)`
Syntax: `(NP-(DET-een)(N-sneltrein))`

b. APS: `(sneltrein<def>)`
Syntax: `(NP-(DET-de)(N<def>-sneltrein))`

c. APS: `(sneltrein<def>(P:naar(Amsterdam<prop>)))`
Syntax: `(NP-(DET-de)(N<def>-sneltrein)`
`(PP-(P-naar)(NP-(N<prop>-Amsterdam))))`

[4] More accurately, Base is the transducer generated by the AGFL system from the grammar that defines all and only APS's.

[5] Some syntactic feature are omitted for the sake of presentation.

```
d.  APS:     (V:nemen[<listen>]
             [sneltrein<def>(P:naar(Amsterdam<prop>))])
    Syntax:  (VP-(NP<arg>-(N<listen>-))
             (NP<arg>-(DET-de)(N<def>-sneltrein)
             (PP-(P-naar)(NP-(N<prop>-Amsterdam)))))
```

The grammar fragment in Figure 6.1 shows how this can be accomplished in AGFL.[6] Notice how the square brackets of an APS node give rise to the insertion the feature `arg` at the phrasal GRAMTSY node. Of course, this grammar only illustrates the principle; the actual implementation employs the affix level for concise formulation, takes care of the notational conventions, and takes into account the additional tasks of Base described below.

6.2.3.2 Adding purely syntactic information

The second task of Base concerns adding purely syntactic information. Some properties of words are purely syntactic: they affect the syntactic structure, but make no semantic contribution. For example, the neuter/non-neuter distinction in common nouns affects the choice of the determiner: a singular, definite, *neuter* noun takes the determiner *het* (e.g. *het perron* 'the platform'), whereas other definiteness take *de* (e.g. *de trein* 'the train' *de perrons* 'the platforms'). It belongs to the task of Base to add this lexical information to the basic syntactic structure, for example, to attach the feature `neut` to neuter words.

Other purely syntactic information includes:

- the preposition of a verb's prepositional object
- the auxiliary that a main verb takes to form its perfect tense, either *zijn* 'be' or *hebben* 'have' (10)
- whether the complementizer of a finite sentential object of a verb is *of* rather than the default *dat* (11)
- whether an infinite sentential object does *not* take the default complementizer *om* (12)
- whether a noun is non-countable (mass noun)

(10) a. hij heeft gereden
he has driven

b. hij is overgestapt
he is transfered

[6] If a rule has no transduction part (cf. `MajorLemma`), the AGFL system assumes that its transduction is the concatenation of the transductions of its right-hand symbols, and that the transduction of a terminal is the terminal itself.

```
MajorCategoryNode :
    "[",
        MajorLabel,
            MajorLemma,
               Features,
                   Daughters,
                      "]" /
    "(",
        MajorLabel,
            "P<arg>-",
                Daughters,
                    "(",
                        MajorLabel,
                            Features,
                               "-",
                                  MajorLemma,
                                      ")",
                                          ")" ;
    ...

MajorLabel :
    "N:" / "N" ;
    "V:" / "V" ;
    ...

MajorLemma :
    "bloem" ;
    "trein" ;
    ...
```

Figure 6.1: AGFL grammar that shows the principle for projecting the basic syntactic structure.

(11) a. zij zegt dat de sneltrein gaat
she says that the fast-trein goes

b. zij vraagt of de sneltrein gaat
she asks if the fast-train goes

(12) a. zij belooft (om) te werken
she promises comp to work

b. zij beweert te werken
she declares to work

SynR assumes reasonable default values for unknown words. For instance, it is assumed that the default complementizer for a finite sentential object is *dat* (cf. (11-a)) rather than *of*, because this is the most frequent form. As a result, only exceptional words have to be added to the Base lexicon.

6.2.3.3 Adding the structure of compound words

For the purposes of pitch accent placement, the internal structure of compound words with two pitch accents ('even stress') needs to be known, as will be explained in Section 7.3.1. Moreover, the particle and the verbal part of a compound verb must be accessible for syntactic realization. Some examples are given in (13).

(13) a. APS: (N:Utrecht CS)
Syntax: (N<prop>-(N-Utrecht)(N-centraal))
String: Utrecht Centraal
Utrecht Central

b. APS: (V:aankomen)
Syntax: (V<zijn,pred>-(P-aan)(V-komen))
String: aankomen
arrive

c. APS: (A:donkerbruin)
Syntax: (A-(A-donker)(A-bruin))
String: donkerbruin
dark-brown

d. APS: (ADV:altijd)
Syntax: (ADV<lrr>-(ADV-al)(N-tijd))
String: altijd
always

The feature lrr in (13-d) is added to trigger the Lexical Rhythm Rule during pitch accent placement. This is a deaccentuation rule that, depending on the context, deletes one of the accents on a word like *altijd*, which belongs to a lexically marked class of exceptions. This will be further discussed in Section 7.3.4).

6.2.3.4 Expanding numbers and times

In order to be pronounced, Numerals have to be expanded to phrases first. Some examples are given in (14). There is quite some irregular behaviour in the mapping. Although we will not go into the implementation, all of it can be conveniently captured in an AGFL. Notice that two digit numbers may be expressed by a single word (14-b,d). How much structure is generated is determined by how much is needed for the purpose of SLG, to be precise, for accent placement. In (14-e), for instance, Base generates the internal structure of the compound, because it may carry two accents. In contrast, no internal structure is generated in (14-c), even though *vijftien* is a compound from a morphological point of view. The same goes *eenen* in (14-e).

(14) a. (NUM:1) ⟶ (NUMP-(NUM-een))
b. (NUM:11) ⟶ (NUMP-(NUM-elf))
c. (NUM:15) ⟶ (NUMP-(NUM-vijftien))
d. (NUM:50) ⟶ (NUMP-(NUM-vijftig))
e. (NUM:51) ⟶ (NUMP-(NUM-(NUM-eenen)(NUM-vijftig)))

A similar concern regarding structure generation is visible in the expansion of larger numbers. In (15-a), the feature urr is introduced, which will later play a role during pitch accent placement, and is added to *honderd* 'hundred', *duizend* 'thousand', *miljoen* 'million', etc. (15-b) shows that *twee* is taken to be a premodifier of *honderd*, comparable to an adjective. However, *honderd* and *vijftig* are assumed to be conjuncts in a coordinated numeral phrase in (15-c), because a conjunction can be inserted between them: we can say *honderd en vijftig*.[7]

(15) a. (NUM:100) ⟶ (NUMP-(NUM<urr>-honderd))
b. (NUM:200) ⟶
(NUMP-(NUMP-(NUM-twee))(NUM<urr>-honderd))
c. (NUM:150) ⟶
(NUMP<conj>-(NUMP-(NUM<urr>-honderd))
(NUMP-(NUM-vijftig)))

[7] The same analysis might be applied to (14-e): *een en vijftig*. However, there are some differences. First, the conjunction cannot be omitted. Second, the conjunction is subject to resyllabification, and reduction to schwa: /e-nən-vɛif-təx/. Applying the same phonological processes to (15-c) makes it sound Belgian Dutch rather than Standard Dutch: /hɔn-dər-tən-vɛif-təx/. Our phonological phrasing algorithm (Section 3.3.3.1) predicts a φ-boundary after *honderd* in (15-c) (but not in (14-e)). Since the domain for resyllabification is assumed to be the Phonological Word, a φ-boundary blocks resyllabification. Thus, the observed difference is in accordance with our predictions.

d. (NUM:151) ⟶
(NUMP<conj>-(NUMP-(NUM<urr>-honderd))
(NUMP-(NUM-(NUM-eenen)(NUM-vijftig))))

e. (NUM:251) ⟶
(NUMP<conj>-(NUMP-(NUMP-(NUM-twee))
(NUM<urr>-honderd))(NUMP-(NUM-(NUM-eenen)
(NUM-vijftig))))

A drawback of the above APS format for numerals is that we cannot indicate a non-default focus distribution, which is required in counting. Consider the series in (16).

(16) twintig, eenentwintig, tweeentwintig, drieentwintig, ...
twenty one-and-twenty two-and-twenty three-and-twenty
'20, 21, 22, 23, ...'

Only the first occurrence of *twintig* is accented, presumably because it is unfocused (given/predictable) in the subsequent occurrences. To accommodate this fact, there is a more elaborate way to specify numbers in an APS using the feature back to indicate that a part of the numeral is backgrounded.

(17) (NUM:(D:2<back>)(D:1)) ⟶
(NUMP-(NUM-(NUM-eenen)(NUM<back>-twintig)))

All APS's for numerals can be expressed in this format; the shorter format introduced earlier is actually only an abbreviation for this more elaborate specification when all parts are focused.

However, this allows some rather puzzling APS's. Should there be any difference in the realization of the following pair?

(18) a. (NUM:(D:1)(D:1<back>)) ⟶ (NUMP-(NUM<back>-elf))

b. (NUM:(D:1<back>)(D:1)) ⟶ (NUMP-(NUM<back>-elf))

Notice that although the focus distribution in the APS's is different, both result in exactly the same output, and there is no syntactic or prosodic way to express the difference in focus, if there is any. For practical purposes, we assume that both APS result in a backgrounded Numeral.

Finally, Base also expands expressions of time (19). Their syntactic structure naturally fits NP's on the assumption that (19-b) is a reduced form of *acht uur en dertig minuten* 'eight hours and thirty minutes'.

(19) a. APS: (TIME:0800)
Syntax: (NP<time>-(NUMP-(NUM-acht))(N-uur)))
String: acht uur
eight hour

b. APS: `(TIME:0830)` ⟶
Syntax: `(NP<time,conj>-(NP-(NUMP-(NUM-acht))`
`(N-uur))(NP-(NUMP-(NUM-dertig))))`
String: acht uur dertig
eight hour thirty

6.3 The **Trans** component

6.3.1 GRAMTSY

GRAMTSY (*GRAMmaticaal Transformationeel Systeem* 'Grammatical Transformational System') is a program to transform trees by means of transformational rules (Coppen 1990; Coppen 1991). The trees are labelled-bracket structures enhanced with feature structures (cf. (8)). An example of a transformational rule is the following.

```
*--------------------------------------------------------------*
%Rule:      Subject-verb-agreement
--> SD:     ... (VP (NP<subj>1 ... ) ... # (V<fin> === ) # )
--> COND:   f2 = (f1 intersection "<sing,plur>")
--> SC:     #1 #2 ==>
            #1
            #2<&f2>
*--------------------------------------------------------------*
```

The name of this rule is *Subject-verb-agreement.* Its *Structural Description* (SD) describes a node pattern: a VP that contains a subject NP and an finite V. The symbol `...` matches an arbitrary substring, thus `(NP ... )` is the pattern for a NP containing any number of nodes.[8] The symbol `===` stands for a lemma, thus `(V<fin> === )` describes a terminal V. The number `1` after the feature bundle `subj` causes the NP's features – including `subj` – to be assigned to the variable `f1`. Its intersection with the features `sing` and `plur` is assigned to the variable `f2` in the *Condition* part (COND). If the SC matches the input and the COND succeeds, the *Structural Change* (SC) is applied. The SC refers to the tab strings in the SD delimited by hash symbols. The symbol `#1` refers to the first tab string, i.e. `... (VP (NP<subj>1 ... ) ...`; `#2` to the second, i.e. `(V<fin> === )`. The change involves adding the value of `f2` to the second tab.

The effect of this rule is that the features for number occurring on the subject NP are copied to the finite verb. Applying the rule to (20-a) gives (20-b).

(20) a. `(S-(VP-(NP<subj,sing>-(N-u))`
`(NP-(DET-de)(N-trein))(V<fin>-nemen)))`

[8] GRAMTSY balances the brackets in an SD.

b. `(S-(VP-(NP<subj,sing>-(N-u))`
`(NP-(DET-de)(N-trein))(V<fin,sing>-nemen)))`

The next rule moves the verb from within the VP to become a direct daughter of the S node. Applying it to (20-b), we obtain (21).

```
*-------------------------------------------------------------*
%Rule:       Verb-second
--> SD:      (S # (VP ... # (V === ) # )
--> COND:    None
--> SC:      #1 #2 #3 ==>
             #1
             #3
             #2
*--------------------------------------------------------------*
```

(21) `(S-(V<fin,sing>-nemen)(VP-(NP<subj,sing>-(N-u))`
`(NP-(DET-de)(N-trein))))`

For each rule, we can specify whether it must be applied once or multiple times, whether it is optional or obligatory, and whether it may be applied onto its own output. Sets of rules can be collected in grammars and subgrammars, which can be applied cyclically, e.g. to S nodes, in a top-down or bottom-up manner.

The above rules are for expository reasons only, and show only the basic operations in GRAMTSY. The actual rules of Trans are considerably more complex, and we will not describe them in detail. However, it will be clear from this short introduction that GRAMTSY is very suitable for enforcing grammatical constraints and determining word order.

6.3.2 Focus distribution

The focus distribution of an utterance depends on its purpose and context (Section 2.5.1). Since the text planner in an NLG system has access to the communicative goals, the discourse history, and the user model, we assume that the focus distribution is determined by PlanR: it assigns the feature `foc` to every focused word in an APS. However, a default assumption regarding focus is used to reduce the size of APS's: by default, content words are focused, but function words are not. This default is implemented in Trans. As a result, PlanR needs to assigns only non-default focus using the features `foc` and `back`.

So far, only words are focused or backgrounded. However, to apply the accentuation rule SAAR in ProsR, it must be known whether an argument – or rather a syntactic phrase functioning as an argument – is focused. For this purpose, Trans implements the focus projection rules to be described in Section 7.3.1.2. Their effect can be summarised roughly as: a major phrase is focused if its head is a focused content word.

6.3.3 Linking arguments to syntactic functions

The grammatical function of subject is defined by both word order and word form phenomena. In a main clause, the subject occurs adjacent to the finite verb[9]; in a subordinate clause, directly after the complementizer. Word order will be described in Section 6.3.7. In addition, a finite verb must agree with its subject for number and person. This subject-verb agreement will be discussed in Section 6.3.6. Finally, the subject carries nominative case, which surfaces if it is a pronoun. Insertion of the pronoun with the right grammatical case belongs to the tasks of Flex, and will be described in Section 6.4. Here we will focus on the question how the subject is determined.

The argument structure of a main verb consists of up to three arguments, and is encoded in the lexicon. Following modern syntactic theories (Jackendoff 1990; Grimshaw 1990, ch. 11), we assume that the arguments are *linked* to syntactic functions. By default, the first argument is linked to the subject, i.e. it receives the feature subj triggering the corresponding word form and word order. Analogously, it is assumed that the second argument is linked to the grammatical function of direct-object when only two arguments are available, whereas the second argument is linked to the indirect-object, and the third to that of direct-object when three arguments are available. It will be clear that the mutual order of a verb's arguments in an APS is significant, because it determines the default mapping from arguments onto syntactic functions. However, certain grammatical alternations like passive voice or ergative may force another argument to be linked as subject, or even prevent subject linking.

6.3.3.1 Voice

Trans is capable of realizing grammatical voice, i.e. the alternation between the active (22) and the passive form (23). Passive formation is triggered by the feature pas on the verb. As a result, the first argument is degraded to a non-argument PP with preposition *door* 'by', whereas its second argument is linked to the subject function. The feature pas also triggers the insertion of a passive auxiliary by Flex (Section 6.3.4).

(22)	APS:	(V:kweken[tuinder<plur>][plant<def>])				
	String:	tuinders	kweken	de	plant	
		market-gardeners	grow	the	plant	

(23)	APS:	(V:kweken<pas>[tuinder<plur>][plant<def>])				
	String:	de	plant	wordt	door	tuinders	gekweekt
		the	plant	is	by	market-gardeners	grown

If the first argument has no lemma or features, the *door*-PP is omitted (24). Notice, however, that the empty argument must be present in the APS anyway,

[9] There are a few exceptions, which we will ignore here. See Haeseryn et al. (1997, Section 21.4).

because otherwise the *second* argument cannot be identified as such.

(24) APS: (V:kweken<pas>[][plant<def>])
String: de plant wordt gekweekt
the plant is grown

6.3.3.2 Dative shift

Another alternation, often referred to as *dative shift*, involves switching around the second and third arguments. It is triggered by the feature ds, and causes the second argument to be transformed to a PP with default preposition *aan* 'to'[10].

(25) APS: (V:geven[nerven<def,plur>][blad<def>] [kleur(rood)])
String: de nerven geven het blad een rode kleur
the veins give the leaf a red colour

(26) APS: (V:geven<ds>[nerven<def,plur>][blad<def>] [kleur(rood)])
String: de nerven geven een rode kleur aan het blad
the veins give a red colour to the leaf

6.3.3.3 Imperative and non-finite clauses

The imperative has normally no overt subject (27). In terms of linking theory, this may be interpreted as the first argument not being linked to a grammatical function, but staying implicit in the interpretation of the clause. Under this analysis, the imperative is similar to a passive without a *door*-PP. In both cases, the first argument remains unlinked and the verb takes an appropriate conjugation. The difference is that the imperative has no auxiliary support, and the first argument cannot surface as a PP.[11]

(27) APS: (kweken<imp>[][plant<def>](P:in [aarde(A:vochtig)]))
String: kweek de plant in vochtige aarde
grow the plant in moist soil

Non-finite clauses have no overt subject either (28). Whether this is because it is a phonetically empty pronoun (PRO) as in transformational grammar (Haegeman 1991, ch. 5), or because it is directly linked to an argument of the matrix clause as in HPSG (Pollard and Sag 1994) is not relevant with respect to generation. That is, the APS in (28-a) does not have to express the fact that the first argument of *proberen* and the first argument of *kweken* refer to the same person.

[10] If the verb carries the feature voor, the preposition *voor* 'for' is used.

[11] Although sometimes the subject may be linked as well, as in *Wees jij maar stil*, literally 'Be you better quiet' (You better keep quiet).

(28) APS: (V:proberen[<speak>][V:kweken[][plant<def>]])
String: ik probeer de plant te kweken
I try the plant to grow
'I try to grow the plant'

Of course, this coreference is an important fact during the *construction* of an APS, as we want to express the intended meaning. In (29), for example, the first argument of *kweken* is coreferential with the *second* argument in the matrix clause, i.e. it is the listener rather than the speaker who has to grow the plant. This is thus a lexical property of the controlling verb (*proberen* versus *adviseren*) that important during the construction of an APS, but is no longer relevant during the syntactic realization of the APS. Since the construction of APS's is a matter of lexical choice and aggregation, and therefore a part of the microplanning process, it is beyond the scope of the current work.

(29) APS: (V:adviseren[<speak>][<listen,red>]
[V:kweken[][plant<def>]])
String: ik adviseer je de plant te kweken
I advise you the plant to grow
'I advise you to grow the plant'

Therefore, Trans treats non-finite clauses in a simple yet adequate way. If it detects an embedded clause of which the matrix clause carries the feature te (added by Base), it deletes the first argument in the embedded clause.

6.3.4 Auxiliaries

The eight tenses of Dutch described in Haeseryn et al. (1997, p. 113) can be specified in terms of the features past, fut (future), and perf (perfect). Trans inserts the lemmas of the appropriate auxiliaries, a combination of *zijn* 'be', *hebben* 'have', and *zullen* 'will', before the content verb. Conjugation of auxiliaries and the main verb – if required – is left to Flex.[12]

(30) APS: (V:werken<past>[<masc>])
String: hij werkte
he worked

(31) APS: (V:werken<perf,past,fut>[<masc>])
String: hij zou hebben gewerkt
he would have worked

Besides auxiliaries of tense, the auxiliary *worden* is inserted in the case of a passive.[13]

[12] An alternative realization of 2 is *hij had zullen werken*. As there is no apparent difference in meaning, only the first variant is generated

[13] Unless perfect tense is specified, in which case *zijn* functions as the auxiliary of both passive voice and perfect tense. For example, *de plant is gekweekt* instead of **de plant is gekweekt geworden*.

The word *te* 'to' occurs in the verb cluster of non-finite clause, and has a certain similarity to an auxiliary. It is therefore inserted by the same set of rules that insert auxiliaries.[14]

6.3.5 Sentence projection

Recall that Base expands every V in an APS to a VP. Depending on its context, Trans may in turn projects the VP to a sentence. There are three types of sentences: root, bound, and relative. If used adjectivally, the VP is not projected to an S.

Root sentence If the VP is not embedded in another phrase, Trans assumes that a root clause must be generated, and inserts an S node directly dominating the VP node.[15] The word order of a root sentences will be discussed in Section 6.3.7.

Bound sentence If the VP is directly dominated by another VP, Trans generates a bound sentence by inserting an S node with feature `bound`. It checks whether the VP contains a C node, which stems from the explicit specification of a complementizer in the APS. If so, this C node becomes the leftmost daughter of the S node. Otherwise, Trans chooses a default complementizer according to the following logic.[16]

- If the verb of the dominating VP has the feature `noom` and the verb of the embedded VP has the feature `inf`, no complementizer is inserted (cf. example (12-b)).
- If the verb of the dominating VP lacks the feature `noom` and the verb of the embedded VP has the feature `inf`, the complementizer *om* 'to' is inserted (cf. example (12-a)).
- If the verb of the dominating VP has the feature `of` and the verb of the embedded VP lacks the feature `inf`, the complementizer *of* 'if/whether' is inserted (cf. example (11-b)).
- If the verb of the dominating VP lacks the feature `of` and the verb of the embedded VP lacks the feature `inf`, the complementizer *dat* 'that' is inserted (cf. example (11-a)).

The features `noom` ('no *om* complementizer') and `of` are lexically specified and added by Base (cf. Section 6.2.3.2).

[14]Modal auxiliaries may be accounted for in the same way as auxiliaries of tense, which means that they are inserted on the basis of a set of modality features. The same approach may be explored for aspect. This is not yet implemented in SynR.

[15]This default can be overruled by adding the feature `bound` to the VP's verb.

[16]In anticipation of a feature-based choice of all complementizers, Trans adds only the features for a certain complementizer, and leaves the insertion of the word form to Flex (Section 6.4.2)

Relative sentences If the VP is directly dominated by NP and its V does not bear the feature `part`, Trans generates a relative sentence by inserting an S node with feature `rel`. Insertion of relative determiners will be discussed in Section 6.3.7.2.

Adjectivally used verbs There is a strong parallel between the adjective *verzamelend(e)* 'collected' in the (32) examples and the verb *verzamelen* 'collect' in the relative clause examples in (33). The former is often regarded as an adjective, because the suffix *-e* can be attached to it. However, the unsuffixed form is identical to the present participle (*verzamelend*) or past participle (*verzameld*) of the corresponding verb.[17] This suggests that the participle form of most verbs can also be used as an adjective.

The feature `part` (participle) on the V's in (32) prevents projection of their VP's to sentences. By implication, they are not moved to the right of the NP's head, as only S's and PP's are (cf. Section 6.3.7.5). The feature also triggers insertion of the participle form by Flex. Attachment of the suffix *-e* follows from the same rules that account for the suffixation of real adjectives (see Section 6.3.6). As a result, the feature `part` triggers an alternation between verbs used relatively and adjectivally akin to the way `pas` triggers the active-passive alternation. As matter of fact, the two can be used simultaneously, as (32-c-d) shows.[18]

(32)
a. APS: (man<def>(V:verzamelen<part>[] [plant<plur>]))
String: de planten verzamelende man
the plants collecting man

b. APS: (man<def>(V:verzamelen<part>[][]))
String: de verzamelende man
the collecting man

c. APS: (plant<def,plur>(V:verzamelen<part,pas> [man<def>][]))
String: de door de man verzamelde planten
the by the man collected plants

d. APS: (plant<def,plur>(V:verzamelen<part,pas> [][]))
String: de verzamelde planten
the collected plants

---

[17] The present participle occurs as a real verb in only a few constructions (Haeseryn et al. 1997, p. 107). The past participle is used in the passive voice and perfect tense.

[18] In the APS's, empty arguments at the right can be omitted. Thus, (plant<plur>(V:verzamelen<part,pas>[][])) can be shortened to (plant<plur>(V:verzamelen<part,pas>)).

(33) a. APS: (man<def>(V:verzamelen[][plant<plur>]))
String: de man die planten verzamelt
the man that plants collect
'the man that collects plants'

b. APS: (man<def>(V:verzamelen[][]))
String: de man die verzamelt
the man that collects

c. APS: (plant<def,plur>(V:verzamelen<pas> [man<def>][]))
String: de planten die door de man worden verzameld
the plants that by the man are collected

d. APS: (plant<def,plur>(V:verzamelen<pas>[][]))
String: de planten die worden verzameld
the plants that are collected

A problem with this approach is that some verbs do not allow a passive (34-a), but can nevertheless be used adjectivally (34-b)

(34) a. *In Amsterdam wordt aangekomen (door de trein)
In Amsterdam is arrived (by the train)

b. de aangekomen trein
the arrived train

To accommodate for these cases, we assume that (34-b) is related to the perfect tense form in (35).

(35) de trein is aangekomen
de train has arrived

With respect to an APS, this means that part can be combined with perf.

(36) APS: (trein<def>(V:aankomen<part,perf>[][]))
String: de aangekomen trein
the arrived train

The corresponding relative form is (37).

(37) APS: (trein<def>(V:aankomen<perf>[][]))
String: de trein die is aangekomen
the train that has arrived

It follows that <part,perf> can also be used with a verb that does allow a passive. In this case, however, the parallel between adjectivally and relatively used verbs no longer holds: the adjective form in (38) clearly means something different than the relative form in (39). We leave this exception for future research.

(38) APS: (man<def>(V:verzamelen<part,perf>[][]))
String: de verzamelde man
the collected man

(39) APS: (man<def>(V:verzamelen<perf>[][]))
String: de man die heeft verzameld
the man that has collected

6.3.6 Agreement

We will refer to the following (and similar) phenomena as *agreement*.

- *Determiner-noun agreement*
  In (40-a), the noun *gebied* is definite-plural, and therefore gets the determiner *de*. In (40-b), it is definite-singular-neuter, and therefore the determiner is *het*. In (40-c), it is indefinite-singular, and the determiner is *een*.

- *Adjective-noun agreement*
  In (40-b), the noun is indefinite-singular-neuter, and therefore the adjectives *groot* and *droog* receive no suffix. For all other combinations of these grammatical properties, the the suffix *-e* is attached (cf. (40-a,c)).[19]

- *Noun-relative determiner agreement*
  A similar relation exists between the noun and the demonstrative determiner introducing the relative clause, which has traditionally been called a 'relative pronoun'. Singular neuter referents take *dat* (40-b,c), all others take *die* (40-a).

- *Subject-verb agreement*
  Finally, the verb *liggen* is plural in (40-a), because its subject *die* is plural, but singular in (40-b,c), because *dat* is singular. Notice that since the subject of the relative clause depends on the noun, it mediates the relation between the noun and the verb.

(40) a. de grote en droge gebieden die onder een bergtop
the large and dry areas that under a mountain-top
liggen
lie

b. het grote en droge gebied dat onder een bergtop ligt
the large and dry area that under a mountain-top lies

[19] This is an oversimplification, as NP's like *een waarnemend minister* 'the acting minister' show. For an extensive discussion, see Haeseryn et al. (1997, ch. 6.4).

c.	een	groot	en	droog	gebied	dat	onder	een	bergtop	ligt
	a	large	and	dry	area	that	under	a	mountain-top	lies

It belongs to the tasks of Trans to enforce agreement of features between nodes, whereas the corresponding insertion of function words (e.g *de*) and inflection of content words (e.g. *ligt*) is the responsibility of Flex.

The APS for (40-a) is (41), and the corresponding tree generated by Base is given in the first part of Figure 6.2. Notice that Base adds neut to the features of *gebied*.

(41)
```
(gebied<def,plur>(A:{(A:groot)(A:droog)})
(V:liggen[<top>][P:onder[bergtop]]))
```

The corresponding tree generated by Trans is given in the second part of Figure 6.2. For the time being, we can ignore the addition of an S-node and the change in word order; we focus on the features here. The noun *gebied* has a feature infl, which indicates that its grammatical properties (<def,neut,plur>) require adjectives to be suffixed. This feature is copied onto its AP premodifier. This happens to be a coordinated structure, and the feature is therefore copied to each of its conjuncts. Finally, it is copied from each AP to its head A.

In a similar way, <def,neut,plur> is copied onto the first determiner, <neut,plur> is copied onto the second determiner, and plur onto the verb. The features resulting from intermediary steps were omitted for expository reasons. Since all features of the noun *bergtop* are default (i.e. singular, indefinite, non-neuter), there are no features to copy to its determiner either.

The enforcement of agreement is implemented as a grammar of transformational rules which traverse the tree in a top-down manner. The rules copy the relevant features from one node onto the next. They form an intricate system that accounts for agreement in all syntactic configurations that SynR can generate, including coordinated structures. Discussing their implementation is beyond the scope of the current work.

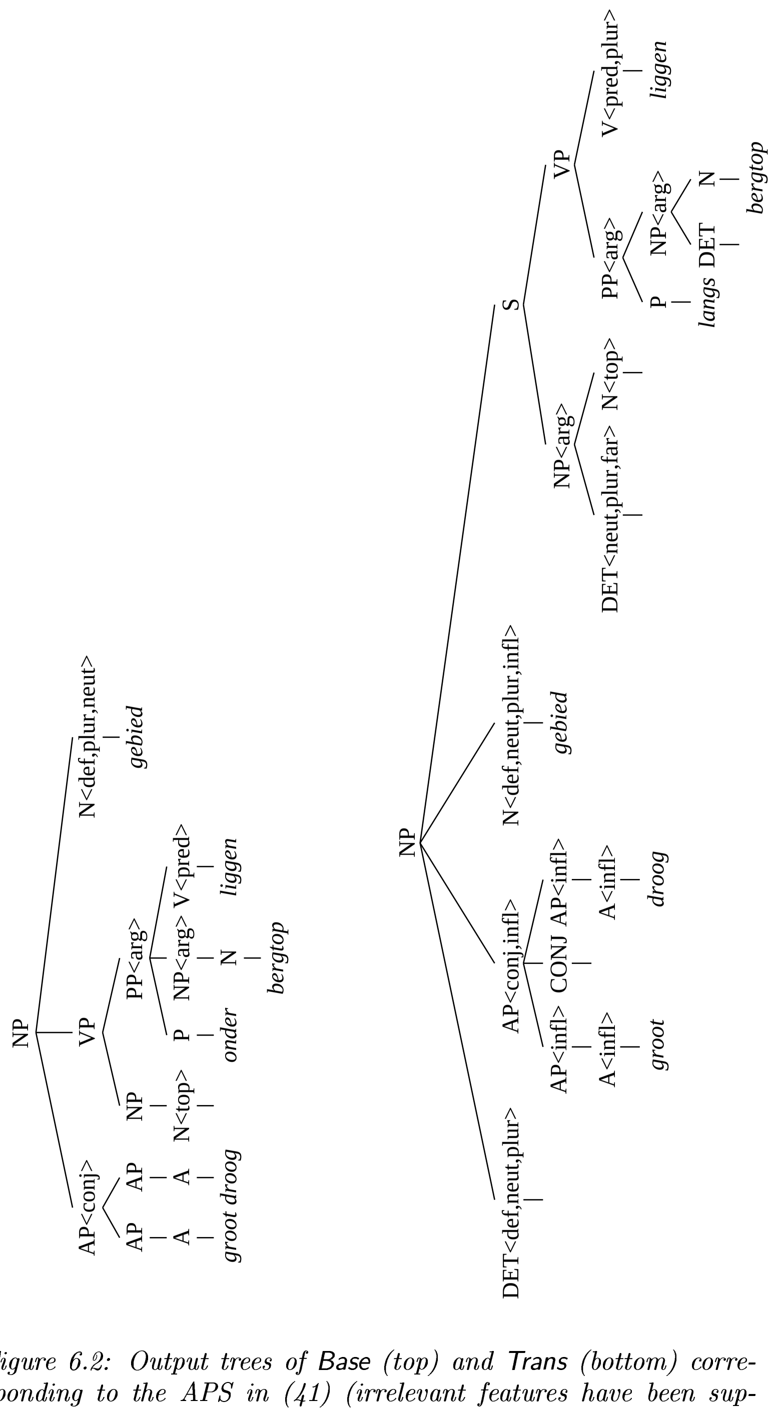

Figure 6.2: Output trees of Base *(top) and* Trans *(bottom) corresponding to the APS in (41) (irrelevant features have been suppressed).*

6.3.7 Word order

6.3.7.1 Verb second

The difference between a root clause and a bound clause is – among other things – reflected in the word order: the former requires movement of the finite verb to the second position in the clause, the so-called *verb second* (V2) transformation (a simplified implementation of this rule was given in Section 6.3.1). In terms of Trans: if an S node has no feature bound or rel, the leftmost verb in the VP is moved to become the leftmost daughter of the S node (42-a).[20]. In a bound or relative clause, the base position of a finite verb is unaffected (42-b).

(42) a. APS: (V:nemen<perf>[<speak>][trein<def>])
Syntax: ($_{S}$ ($_{NP}$ ik) ($_{V}$ heb) ($_{VP}$ de trein genomen))
I have the train taken

b. APS: (V:nemen<perf>[<speak>][trein<def>])
Syntax: ($_{S}$ ($_{C}$ dat) ($_{VP}$ ik de trein ($_{V}$ heb) genomen))
that I the train have taken

6.3.7.2 Topicalisation

A root clause has also a grammatical mood, which is partly reflected in the word order. The default mood is declarative. It requires movement of one the VP's phrasal daughters to the first position of the sentence, which is known as *topicalisation*. By default, the subject will be topicalised (43-a), unless another constituent is explicitly marked with the feature top (43-b).

(43) a. APS: (V:nemen[<listen>][trein<def>])
String: u neemt de trein
you take the train

b. APS: (V:nemen[<listen>][trein<def,top>])
String: de trein neemt u
the train take you

Interestingly, recent syntactic theories claim that topicalisation of a non-subject is different from the fronting of a subject (Zwart 1993); see also Coppen (1991, p. 109) and van Eynde (1997). For instance, only subject clitics (reduced personal pronouns) can appear in fronted positions (44).

(44) a. je hebt me gezien
you have me seen

b. *je heb ik gezien
you have I seen

[20] Movement of *ik* is due to topicalisation; cf. Section 6.3.7.2

There are two other moods that are indicated by a feature on the main verb: `imp` 'imperative' (45-a), and `polar` 'polar question'/'yes-no question' (45-b). Neither of them allows topicalisation. A wh-question – often considered a mood as well (Halliday 1994) – is indicated by the feature `wh` on one of the VP's constituents (45-c-d), and triggers its topicalisation.[21]

(45) a. APS: `(V:nemen<imp>[][trein<def>])`
String: neem de trein
take the train

b. APS: `(V:nemen<polar>[<listen>][trein<def>])`
String: neemt u de trein
take you the train
'do you take the train?'

c. APS: `(V:nemen[<ani,wh>][trein<def>])`
String: wie neemt de trein
who takes the train

d. APS: `(V:nemen[<listen>][<wh>])`
String: wat neemt u
what take you
'what do you take'

Topicalisation also applies to relative clauses, where `top` marks the constituent bound by the noun of the matrix phrase.

(46) a. APS: `(man<def>(V:nemen[][trein<def>]))`
String: de man die de trein neemt
the man that the train takes
'the man who takes the train'

b. APS: `(trein<def>(V:nemen[man<def>][<top>]))`
String: de trein die de man neemt
the train that the man takes

c. APS: `(dag<def>(V:nemen[man<def>][trein<def>]`
`(P:op[<wh>])))`
String: de dag waarop de man de trein neemt
the day at-which the man the train takes
'the day at which the man takes the train'

6.3.7.3 Extraposition

Extraposition is rightward movement of a VP-internal constituent so that it becomes the rightmost daughter of the dominating S. It is normally restricted to PP's and S's (either a bound S or a non-restrictive relative S). Extraposition

[21] This is an oversimplification, as there are also forms like *Weet jij wie de trein neemt?* 'Do you know who takes the train?'.

of a PP is not obligatory, but is preferred if it is 'heavy'. We have made this operational in the following way: a PP is heavy if it contains another PP or an S. Alternatively, extraposition can be forced or blocked by means of the features `heavy` and `light` respectively. Of course, a topicalised PP cannot be extraposed.

Extraposition of S's is the default (47-a), but may be blocked for very light clauses (47-b).

(47) a. APS: (V:nemen<polar,perf>[<listen>] [stoptrein<def>](V:komen(P:in[Amsterdam]) (C:om)))

String: heeft u de stoptrein genomen om in
have you the slow-train taken to in
Amsterdam te komen
Amsterdam to get
'did you take the slow train to get to Amsterdam?'

b. APS: (V:nemen[<listen>][trein<def>(P:naar [Amsterdam])](V:beginnen<light>[] (C:om)))

String: u neemt om te beginnen de stoptrein naar
you take comp to begin the slow-train to
Amsterdam
Amsterdam
'to begin with, take the slow train to Amsterdam'

An NP may be extraposed only if it is extremely heavy, a transformation that is known as *heavy NP shift*. In addition, extraposition can be applied to deeper embedded phrases, in particular to attributively used heavy PP's and to non-restrictive relative clauses.[22] In all these cases, extraposition is applied only if the APS explicitly specifies the feature `heavy`.

6.3.7.4 Scrambling

Determining the order of the remaining VP daughters is known as *scrambling* in transformational terms. This is a notoriously difficult problem, since the order is affected by many, sometimes contradictory requirements. We follow Haeseryn et al. (1997, ch. 21) and Verhagen (1986) in assuming that the word order is affected by syntactic category, syntactic function and focus. There are certainly other factors involved, particularly in the more marked word orders, but these three appear to be the most important. The implementation consists of transformational rules that move constituents until they fit the pattern in (48).

[22]In some cases, however, even restrictive relative clauses may be extraposed: *Ik heb alleen de trein gevonden die naar Arnhem gaat* 'I have only the train found that goes to Arnhem'.

(48) **Word order of VP daughters**

subject
unfocused indirect-NP-object
unfocused direct-NP-object
unfocused indirect-PP-object
unfocused modifiers*
focused modifiers*
focused indirect-NP-object
focused direct-object
focused indirect-PP-object
verbal cluster

Some examples are given in (49).

(49) a. APS: (V:geven[nerf<plur,def>][][<far>]
(ADV:indirect)(C:dat))
String: dat de nerven het dat indirect geven
that the veins it that indirectly give

b. APS: (V:geven<ds>[nerf<plur,def>][][<far>]
(ADV:indirect)(C:dat))
String: dat de nerven dat daaraan indirect geven
that the veins it to-that indirectly give

c. APS: (V:geven[nerf<plur,def>][][kleur]
(ADV:indirect)(C:dat))
String: dat de nerven het indirect een kleur geven
that the veins it indirectly a colour give

d. APS: (V:geven<ds>[nerf<plur,def>][blad]
[<far>](ADV:indirect)(C:dat))
String: dat de nerven dat indirect aan een blad
that the veins it indirectly a colour give
geven

e. APS: (V:geven[nerf<plur,def>][blad][kleur]
(ADV:indirect)(C:dat))
String: dat de nerven indirect een blad een kleur
that the veins indirectly a leaf a colour
geven
give

f. APS: (V:geven<ds>[nerf<plur,def>][blad][kleur]
(ADV:indirect)(C:dat))
String: dat de nerven indirect een kleur aan een blad
that the veins indirectly a colour to a leaf
geven
give

Although this simple approach works well for our application domains, there is room for improvement. For example, (50) is a more natural variant of (49-b) which we cannot account for.

(50) dat de nerven dat daar indirect aan geven
that the veins it that indirectly to give

Clitics violate the word order pattern in (48), because the direct object appears *before* the indirect object.

(51) ik heb 't 'm gegeven
I have it him given
'I have given it to him'

For another interesting problem, consider the pair in (52). In (52-a), the possessive determiner *zijn* 'its/his' refers to the preceding indirect object *een blad*. However, if *een blad* follows *zijn* as a result of dative shift, the coreference is broken. This is an example of what Postal (1971) called *crossover phenomena*. Apparently, some form of binding theory is required here to account for the interaction between word order and the interpretation of anaphora.

(52) a. dat de nerven indirect een blad zijn kleur geven
that the veins indirectly a leave its colour give
'that the veins indirectly give a leave its colour'

b. ?dat de nerven indirect zijn kleur aan een blad geven
that the nerves indirectly its colour to the leave give

The placement of particles and predicative elements, which appears to be strongly related to the verb cluster (van Dreumel 2000), has not been addressed in SynR either.

Correctly predicting the word order in complex VP's is probably the ultimate challenge for any syntactic realizer for Dutch.

6.3.7.5 Word order in NP's

Compared to the variation in S's and VP's, the word order in NP's is fairly rigid. Determiners, quantifiers, adjectives, verbs used adjectivally, and nominal premodifiers must all precede the head; attributive PP's and relative clauses must follow it. This is accomplished by a number of movement rules that we will not go into.

One interesting issue in relation to focus is the mutual order of adjectives, which appears to be determined by semantic criteria: the more 'inherent' (or 'objective' as opposed to 'subjective') the property expressed by an adjective, the closer the adjective occurs to the noun (van der Lubbe 1958). For instance, a material adjective is assumed to be more inherent than a colour adjective, which is in turn more inherent than a size adjective.

(53) grote groene houten stelen
big green wooden stalks

It is possible to deviate from this order if only one of the adjectives is focused, and therefore accented (Haeseryn et al. 1997, p. 837-840).

(54) groene grote houten stelen
green big wooden stalks

In anticipation of a theoretical account of this matter, we assume that the microplanner determines the order of the adjectives. Therefore, SynR respects the order of the adjectives in the APS.

6.4 The Flex component

The Flex component is responsible for lexical and morphological realization: the insertion of function words and the inflection of content words. It maps the features of a terminal node, which have been collected in the previous two steps, onto an appropriate word. This is again accomplished by an AGFL. The idea is simple yet effective: we parse the syntactic structure generated by Trans and while parsing the terminal nodes, we use its features to transduce the node's content (a lemma or an empty string) to a word.

6.4.1 Inflection of content words

We will explain this process using the inflection of adjectives as an example. Adjectives marked with the feature `infl` normally receive the inflectional suffix *-e* (55-a). In some cases, this changes the spelling (55-b); in other cases (55-c), it affects the pronunciation of the stem as well (/xrɔf/ becomes /xrofə/). Moreover, some adjectives (e.g. most material adjectives) resist suffixation (55-d).

(55) a. groen ⟶ groene
green green+`infl`

b. groot ⟶ grote
big big+`infl`

c. grof ⟶ grove
coarse coarse+`infl`

d. houten ⟶ houten
wooden wooden+`infl`

In addition, adjectives can be marked for degree. We assume that an unmarked adjective is *positive*. The feature `cmp` marks a *comparative*, which normally receives the suffix *-er*, whereas `sup` marks a superlative, which is by default inflected with the suffix *-st* (56-a). However, deviations may occur (56-b,c) to the extent of complete irregularity (56-d).

(56) a. groen, groener, groenst
green, greener, greenest

b. groot, groter, grootst
big, bigger, biggest

c. duur, duurder, duurst
expensive, more expensive, most expensive

d. goed, beter, best
good, better, best

The problem to be solved by Flex is choosing the correct word form on the basis of an adjective's lemma and features.

Given the format of a GRAMTSY node (cf. (8)), it is not difficult to write a grammar that describes GRAMTSY output, and to generate a parser for it. The grammar rules for non-terminals are of no interest; their only function is for traversing the tree down to the terminals. It is the transduction rules for terminal nodes that do the real work. Figure 6.3 shows the rule for an Adjective terminal. It contains a call to another rule for parsing the feature set (AdjectiveFeatureSet), consisting of a list of comma-separated features between angled brackets, which is given in Figure 6.4. The key point to notice is that if relevant features are encountered, they are 'lifted' to the affix level. For instance, if the feature set contains infl, the value of the affix inflection becomes +infl. Likewise, if it contains cmp, the value of the affix degree becomes cmp. If no features are specified, the affixes remain unrestricted.[23]

Suppose the terminal node at hand is (A<infl>-groen). After its features have been parsed, inflection is restricted to +infl, and degree is still unrestricted. Next, the rule in Figure 6.3 attempts to parse the node's lemma by calling Adjective(+infl,degree). The rules for lemma parsing are given in Figure 6.5. For convenience, they are labelled 1a through 2c. According to the AGFL implementation, the rules are tried in the order in which they are given in the grammar. Rule 1a is not applicable, since it requires -infl. The left-hand side of rule 1b matches, but groen is not a member of any of its alternatives. Rule 1c matches, causing the subsequent call Adjective(degree). This matches rule 2a (degree is specialised to pos); the expression $MATCH("[a-z]*") is a regular expression that matches any string of lower case characters. The transduction returned by 2a is the original lemma groen. The transduction of rule 1c attaches suffix e, changing it to groene. This is in turn incorporated in the transduction of Figure 6.3, resulting in the string (A<infl>-groene). Thus, (A<infl>-groen) is transduced

[23]The feature set may contain other features, which are irrelevant for inflection. These are accounted for by a default rule not shown in Figure 6.4. Notice also that the order in which the features occur is irrelevant.

```
AdjectiveTerminalNode :
    "(",
        "A",
            AdjectiveFeatureSet(inflection,compare),
                "\-",
                    Adjective(inflection,compare),
                        ")" /
                            "(A",
                                AdjectiveFeatureSet,
                                    "-",
                                        Adjective,
                                            ")".
```

Figure 6.3: AGFL rule for transducing an adjective terminal node.

to `(A<infl>-groene)`. Some other examples can be traced with the help of Table 6.1.

The same method is applied to inflect other content words:

- nouns are inflected for number (`plur`) and possession (`pos`)
- verbs are inflected for form (`fin`, `part`, or `imp`), number (`plur`), person (`speaker`, `listen`), tense (`past`), perfect (`perf`), and suffix (`infl`).[24]

Adverbs, which have no inflection, are simply ignored.

6.4.2 Insertion of function words

Insertion of function words is essentially the same as inflection, with the exception that there is usually no lemma. Flex inserts and inflects auxiliaries, complementizers, conjunctions, and pronominal adverbs. In addition, Flex inserts a wide and nearly complete range of determiners, including articles and those words that have traditionally been called 'pronouns' (Section 2.6.3). Insertion takes place according to features for number (`plur`), person (`speaker`, `listen`), case (`subj` 'subjective'), definiteness (`def`), neuter (`neut`), animacy and gender (`ani`, `fem`, and `masc`), formality (`inf` 'informal' and `red` 'reduced'), possession (`pos`), space (`near` and `far`), and 'wh-ness' (`wh`).

As explained earlier (Section 6.3.4), Trans inserts the lemmas of auxiliaries of tense and voice. Their inflection is left for Flex, and is accomplished in the same way as content verbs are inflected.

Insertion of complementizers is currently limited to those that are grammatically enforced and neutral in meaning, i.e *dat* 'that', *of* 'whether/if', and *om* (cf. Section 6.3.5). There are many more complementizers, such as those

[24] If used adjectivally; cf. Section 6.3.5

```
AdjectiveFeatureSet(inflection,compare) :
    "<",
       AdjectiveFeatureSequence(inflection,compare),
            ">" /
                "<",
                    AdjectiveFeatureSequence,
                        ">" ;
    .

AdjectiveFeatureSequence(inflection,compare) :
    AdjectiveFeature(inflection,compare),
        ",",
            AdjectiveFeatureSequence(inflection,compare) /
                AdjectiveFeature,
                    ",",
                        AdjectiveFeatureSequence ;
    AdjectiveFeature(inflection,compare) /
        AdjectiveFeature;
    .

AdjectiveFeature(+infl,compare) :
    "infl" /
        "infl".

AdjectiveFeature(inflection,cmp) :
    "cmp" /
        "cmp".

AdjectiveFeature(inflection,super) :
    "super" /
        "super".

# affix grammar
inflection :: +infl; -infl.
degree :: pos; cmp; sup.
```

Figure 6.4: AGFL rules for transducing an adjective's features.

```
# 1a
Adjective(-infl,degree) :
    Adjective(degree) /
        Adjective.

# 1b : irregular positive
Adjective(+infl,pos) :
    "groot" / "grote" ;
    "open" / "open" ;
    "roze" / "roze" ;
    ...

# 1c
Adjective(+infl,degree) :
    Adjective(degree) /
        Adjective,
            "e".

# 2a : regular positive
Adjective(pos) :
    $MATCH("[a-z]*") /
        $MATCH.

# 2b: comparative
Adjective(cmp) :
    "groot" / "groter" ;
    "donker" / "donkerder" ;
    "goed" / "beter" ;
    "kwaad" / "erger" ;
    ...
    $MATCH("[a-z]*") /
        $MATCH,
            "er".

# 2c: superlative
Adjective(sup) :
    "goed" / "best" ;
    "kwaad" / "ergst" ;
    ...
    $MATCH("[a-z]*") /
        $MATCH,
            "st".
```

Figure 6.5: AGFL rules for transducing an adjective's lemma, to its inflected form.

Table 6.1: Input-output samples of adjective inflection: Input *is a adjective terminal node;* Rules *are the rules in Figure 6.5 applied during the transduction;* Word *is the word form in the output node.*

Input:	*Rules:*	*Word:*
`(A<>-groen)`	1a & 2a	`groen`
`(A<infl>-groen)`	1c & 2a	`groene`
`(A<cmp>-groen)`	1a & 2b	`groener`
`(A<infl,sup>-groen)`	1c & 2c	`groenste`
`(A<infl>-groot)`	1b	`grote`
`(A<cmp>-groot)`	1a & 2b	`groter`
`(A<infl,sup>-groot)`	1c & 2c	`grootste`
`(A<>-goed)`	1a & 2a	`goed`
`(A<cmp>-goed)`	1a & 2b	`beter`
`(A<sup>-goed)`	1a & 2c	`best`

of time (*voordat* 'before'), of reason (*omdat* 'because'), of condition (*tenzij* 'unless'), of concession (*hoewel* 'although'), etc. Although rarely required in our application domains, these can be generated by explicitly including their lemma in an APS. However, if a suitable description of their meaning in terms of semantic and discourse features were available, this could easily be implemented in Flex.

The same goes for the insertion of conjunctions, which is currently limited to *en* and *of*. Since a description in terms of features is lacking, conjunctions other than these must be explicitly specified in the APS.

Finally, Flex also inserts the so-called pronominal adverbs: *er* 'there', *hier* 'here', *daar* 'there', *waar* 'where'. These can be a combined with a preposition: *door* 'through' plus *daar* 'there' becomes the adverb *daardoor* 'because of that' (lit. 'there-through'). In some cases, the combination changes the form of the preposition: *met* 'with' plus *daar* 'there' becomes *daarmee*. Discussing the implementation of these phenomena would lead us to far from the main topic. Suffice it to say that all of these are correctly generated by SynR.

6.5 Discussion

6.5.1 APS's versus templates

The easiest way to generate text is using string templates. In principle, we can generate travel descriptions by filling the slots of templates like (57) with travel parameters. All processing occurs at the string level, without any syntactic, semantic, or conceptual level of representation.

(57) `take the <conveyance> from <departure> to <arrival>`

Why then bother using abstract phrase specifications and a syntactic realizer? Below, we discuss three arguments in favour of using more abstract and general NLG techniques (Reiter 1995; Reiter and Dale 1997).

To begin with, the maintainability of template-based systems decreases if the required variation in the output increases. Since templates allow only limited abstraction and generalisation, generating many different texts requires adding many different templates. The resulting redundancy makes the system large and difficult to maintain. Altering the content or form of the output may involve changes at many points in the code, which is laborious and prone to error.

Another argument, which – to the best of our knowledge – has not yet been explicitly mentioned in the literature, is that a template-based realizer cannot provide the syntactic information that is required for prosodic realization. It is of course possible to annotate templates with prosodic mark-up codes, but this requires expert knowledge: not all programmers who implement string templates for grammatical expressions in his/her native language can confidently assign pitch accents and prosodic boundaries.

Finally, it is generally accepted that NLG systems using syntactic realizers can produce higher-quality output. This is not so much due to the fact that the output is 'more grammatical' than that accomplished with templates, but rather that the presence of an abstract semantic or syntactic level of representation enables the construction of better texts. To appreciate this point with respect to SynR, we will discuss the abstraction embodied in our APS's, and show how it facilitates microplanning.

As explained earlier, an APS is a representation that abstracts away from certain syntactic aspects, in particular, from (most) function words, the inflection of content words, and word order. From a somewhat different perspective, we can distinguish two types of abstraction. First, it abstracts away from purely syntactic phenomena that have no associated semantics. Examples of these are the form of a definite article (*de* or *het*), the difference in word order between root clauses and bounded clauses, agreement of a finite verb with the subject, and the suffixation of an attributive adjective. These may be considered formal properties of an utterance that facilitate its processing, but do not contribute to its meaning. Second, it abstracts away from syntactic alternations related to perspective and information structure. For instance, the difference between the active and passive form is no more than the feature `pas`. This represents a considerable abstraction of the process of linking the second argument to the subject, linking the first argument to a *by*-PP, inserting an auxiliary, and changing the main verb's conjugation.

These abstractions considerably simplify the task of microplanning, i.e. lexicalisation, referring expression generation, and aggregation. With respect to referring expression generation, the algorithm can concentrate on the question whether an anaphoric expression can be used, given the current discourse history; it does not need to be concerned with the grammatical case of a pronoun. Similarly, it can take definiteness decisions independently of the different syntactic realization of this property.

With respect to aggregation, it allows generalisations that cannot be expressed in terms of surface strings. Consider the example in (58), a standard example of aggregation by conjunction.

(58) a. boeren gebruikten de bladeren als thee
farmers used the leaves as tea

b. boeren gebruikten de bladeren als kleurstof
farmers used the leaves as colour

c. boeren gebruikten de bladeren als thee en als kleurstof
farmers used the leaves as tea and as colour

Suppose these are generated by instantiation of the template `x gebruikten y als z` ‘x used y as z’. We can formulate the following quasi-formal aggregation rule to rewrite (58-a-b) to (58-c).

(59) Given ($\mathtt{x_1\ used\ y_1\ as\ z_1}$) and ($\mathtt{x_2\ used\ y_2\ as\ z_2}$),
if $x_1 = x_2$ and $y_1 = y_2$ then
replace with ($\mathtt{x_1\ used\ y_1\ as\ z_1\ and\ z_2}$)

The same type of aggregation occurs in (60).

(60) a. de bladeren werden door boeren als thee gebruikt
the leaves were by farmers as tea used

b. de bladeren werden door boeren als kleurstof gebruikt
the leaves were by farmers as colour used

c. de bladeren werden door boeren gebruikt als thee en als kleurstof
the leaves were by farmers used as tea and as colour

To account for this, however, we need another aggregation rule that matches the template `(x was used as y)`. Obviously, the two rules are closely related, but using string templates prevents the generalisation from being made.

This problem can be solved by increasing the abstraction of a phrase specification. The APS's corresponding to (58-a) and (60-a) are given in (61).[25]

[25]For the sake of the argument, we assume that *als* is a preposition here. It may as well be analysed as the complementizer of a clause with a gapped verb, i.e. *als thee zijnde* 'as tea being'.

(61) a. (V:gebruiken<past>[boer<plur>][blad<plur,def>]
[P:als[thee<nonspec>]])

b. (V:gebruiken<past,pas>[boer<plur>][blad<plur,def>]
[P:als[thee<nonspec>]])

At this level of abstraction, the only difference between them is the feature pas. The similarity of the APS's allows us to write a generalised aggregation rule.

(62) Given (V:gebruiken[x_1][y_1][z_1])
and (V:gebruiken[x_2][y_2][z_2]),
if $x_1 = x_2$ and $y_1 = y_2$ then
replace with (V:pred_1[x_1][y_1][{[z_1][z_2]}])

Admittedly, this rule can be improved by generalising over the predicate, and we can write another aggregation rule to factor out the preposition *als*. In fact, aggregation is also constrained by pragmatic and contextual factors (Elhadad, McKeown, and Robin 1997; Dalianis and Hovy 1993; Dalianis 1997; Dalianis and Hovy 1997; Shaw 1998). However, aggregation is not the topic of the current work; the point here is that the abstraction of an APS facilitates aggregation.

To sum up, it was argued that using APS's in combination with SynR provides significant advantages compared to string templates, which are essentially due to a higher level of abstraction. First, it is easier to maintain, especially when a system must be scaled up. Second, templates cannot provide the linguistic structure required for prosodic realization in SLG systems. Third, APS's facilitate microplanning – although not carried out in our systems – resulting in higher-quality output.

6.5.2 Syntactic realization and focus

The canonical word order in Dutch, as in many languages, is to have unfocused information precede focused information.[26] This relation between focus and word order is currently only exploited within the VP, but may be extended to clauses. Grammatical alternations like passive and dative shift may be interpreted as the means to obtain the preferred word order, and the same goes for transformations like topicalisation and extraposition. However, focus is clearly not the only reason for applying these operations. Passive, for instance, suggests abstraction and objectivity by suppressing the role of Agent, and is therefore often found in news reports or scientific texts ('It is commonly assumed that ...'). Topicalisation also occurs to front focused information, violating the canonical order in the second clause of the following piece of discourse.

[26] However, adjectives appear to form an exception (cf. Section 6.3.7.5.)

(63) u kunt de trein van 00.30 nemen
u can the train of 00.30 take
'you can take the 00.30 train'

*latere verbindingen zijn er niet
later connection are there not
'there are no later connections'

As focus is not the only consideration in choosing these alternations, albeit an important one, we think that the choice should not be made by the syntactic realizer, but rather by the microplanner.

6.5.3 Comparison with other systems

Work on syntactic realization for Dutch is relatively rare. Probably the best known work is that by Kempen and Hoenkamp (1987) on Incremental Procedural Grammar (IPG), which has been continued by De Smedt (1990, 1996). In contrast to SynR, it aims at a psychologically plausible model of the syntactic aspects of human sentence production. IPG is implemented in Lisp as a set of procedures for generating sentence parts, and a stack that serves as the central data structure for communication. It has no clear distinction between the grammar (data part) and its implementation (procedural part), which is generally considered a disadvantage from a computational linguistic point of view. Other work on syntactic realization for Dutch includes Teunissen (1997), which focuses on a multilingual implementation of Semantic Syntax with limited coverage, and Degand (1993, Degand (1996), which is based on Systemic Functional Linguistics.

The multilingual text generation system KPML (Matthiessen and Bateman 1991), which is a continuation of the PENMAN system for English, has a grammar for Dutch as well (Degand 1993; Degand 1996). It profits from a long tradition of NLG using Systemic Functional Grammar (Halliday 1994), and has a thorough theoretical foundation as well an extensive grammar development environment. The coverage of the Dutch grammar appears to be comparable to that of SynR. The input structures have a slightly higher level of abstraction, closer to a semantic specification, but appear to be less suitable for manual input due to their complex structure.

Another well known surface generator for English is SURGE (Elhadad, McKeown, and Robin 1997; Elhadad, McKeown, and Robin 1997), which is actually a Functional Unification Grammar (FUG) implemented in the FUF system (Elhadad 1993). It is widely used, has a broad coverage, and allows input as case frames or as abstract syntactic structures. In earlier work on syntactic realization, we implemented a FUG for Dutch called SEM2SYN (Marsi 1998b), although with a substantially smaller coverage than SURGE. Its input has the form of a case frame consisting of a predicate, its participants, and a number of circumstances, which is slightly more abstract than SynR's input. In principle, there is a clear separation between the declarative grammar and its processing by FUF. Defaults for underspecified input follow naturally from the unification mechanism. However, a disadvantage of FUF is that it does not support Dutch inflection, which must therefore be coded in the grammar, causing a lot of overhead. In contrast to SynR, SEM2SYN – or rather FUF – uses backtracking during generation. We found that the use of advanced control mechanisms like goal-freezing becomes necessary when generating long sentences to prevent excessive backtracking. Unfortunately, this obscures the separation between grammar and processing, and complicates tracing. In addition, whereas transformational rules provide a natural way of changing word order, this proved to be less naturally expressed in a FUG. In particular, word ordering within VP's gave rise to rather cumbersome formulations.

Although for English, we also mention the commercial RealPro surface generator for English by CoGenTex (Lavoie and Rambow 1997; CoGenTex, Inc 1998), because it is closest to SynR in terms of input and design. The input is a Deep Syntactic Structure (DSynSs) similar to our APS.

(64)
```
help [aspect:cont,polarity:neg
(I John [class:proper-noun]
II Mary [class:proper-noun])
```

'John is not helping Mary'

RealPro has pipelined modules for inserting function words and default features (Deep Syntactic Component), for determining agreement and word order (Surface Syntactic Component), for morphological processing (Deep Morphological Component), and for orthographic processing (Graphical Component). In contrast to SynR, it is based on Meaning-Text Theory (MTT) (Mel′čuk 1988), a form of dependency grammar. It produces strings, but not their syntactic constituent structure.

In sum, none of the above systems offers precisely the kind of functionality that is required in the current work, which is the reason that SynR was built.

6.5.4 Orthography

In most NLG systems, syntactic realization includes generating orthographic properties such as punctuation symbols and capitalisations. In SLG, however, there is no need for orthographic processing, because the output is speech rather than text. Capitalisation of proper nouns, for example, has no equivalent in speech. Admittedly, some punctuation symbols would appear to have prosodic correlates. In read speech, for instance, a comma corresponds to an intonational phrase boundary, and a question mark to a tune ending in a high rise. However, this does not imply that the intonational form is *derived* from the orthographic form, thus requiring orthographic processing in an SLG system. Rather, both the orthographic form and the prosodic form encode aspects of the linguistic structure of a sentence/utterance, its function, and its relation to the context. For example: the writer's/speaker's intention to verify the truth of a proposition by means of a yes/no question results in a question mark in text and in question tune in speech.[27] Likewise, a piece of information that is not an essential contribution according to the writer/speaker may surface between parentheses in text and be pronounced in a lower register in speech. Therefore, the relation between orthographic form and prosodic form is only *indirect*. It may be exploited in TTS, by lack of access to the underlying information, but it should be ignored in SLG, where the relation between the underlying information and the prosodic form can be modelled directly.[28]

6.5.5 Coverage

SynR currently generates a reasonable subset of Dutch. The following lists are intended to give an impression of its coverage. With respect to clauses, it is capable of generating root clauses with different grammatical moods (declaratives, polar questions, wh-questions, imperative), finite and non-finite bound clauses, relative clauses, and expressing passive voice, dative shift, tense, topicalisation and extraposition. As far as NP's are concerned, it can handle determiners (articles, also personal, demonstrative, interrogative, and possessive 'pronouns'), and several types of modifications, such as positive, comparative, and superlative adjectives, attributive PP's, relative S's, numerals and times. In addition, it can process pronominal adverbs and expand numerals. Of course, there are many ways to extend its coverage, for example by generating negation, quantification, modal verbs, appositions, grammatical alternations like ergative, clefts, and pseudo-clefts, and reflexive and reciprocal determiners.

6.6 Summary

This chapter described SynR, a syntactic realizer for Dutch that maps abstract phrase specifications (APS's) into grammatical expressions and their syntactic structures, allowing a user/system to abstract from most of idiosyncrasies of syntax. It provides the functionality required for SLG not provided by any of the other systems currently available for Dutch. An overview of SynR is given in Table 6.2.

APS's allow for a concise specification of phrases thanks to a compact format, the notational conventions introduced, and the assumption of reasonable default values for features. It was argued that using APS's in combination with SynR provides a number of advantages compared to string templates. First, it is easier to maintain, especially when a system must be scaled up. Second, templates cannot provide the linguistic structure required for prosodic realization in SLG systems. Third, APS's facilitate microplanning, resulting in

[27] Of course, there may be additional cues, like word order.

[28] An SLG system may output both text and speech in parallel, so a user can read and listen at the same time. This requires explicit synchronisation between orthographic and prosodic realization.

Table 6.2: Overview of SynR.

Component:	*Tasks:*	*Formalism:*
Base	Projecting the basic syntactic structure Adding purely syntactic information Adding the structure of compound words Expanding numbers and times	AGFL
Trans	Distributing focus Linking arguments to syntactic functions Inserting auxiliaries Projecting sentences Enforcing agreement Arranging word order	GRAMTSY
Flex	Inflecting content words Inserting function words	AGFL

higher-quality output.

The architecture of SynR is highly modular and forms a strict pipeline. The use of the special-purpose grammar formalisms AGFL and GRAMTSY guarantees a high degree of separation between the linguistic grammar (data) and the procedural aspects of its implementation. The strengths of affix grammars were employed in the first and last step, i.e. for mapping APS's onto basic syntactic structures in Base, and for inserting and inflecting words in Flex. At the same time, its weakness was avoided by using a more powerful transformational grammar for enforcing agreement relations and determining word order. As a result, SynR is easy to trace, modify, and extend, is reliable and sufficiently fast, and can be reused in future laboratory NLG and SLG systems.

SynR currently generates a number of syntactic structures covering a substantial part of the core grammar of Dutch. However, more constructions can be added. One of the interesting challenges is the word order in VP's in relation to the focus distribution and other grammatical constraints.